一.基本查询

q – 查询字符串，必须的。

fl – 指定返回那些字段内容，用逗号或空格分隔多个。

start – 返回第一条记录在完整找到结果中的偏移位置，0 开始，一般分页用。

rows – 指定返回结果最多有多少条记录，配合 start 来实现分页。

sort – 排序，格式：sort=<field name>+<desc|asc>[,<field name>+<desc|asc>]… 。示例：（ inStock desc, price asc ）表示先 “inStock” 降序，再 “price” 升序，默认是相关性降序。

wt – (writer type)指定输出格式，可以有 xml, json, php, phps, 后面 solr 1.3 增加的，要用通知我们，因为默认没有打开。

fq – （ filter query ）过虑查询，作用：在 q 查询符合结果中同时是 fq 查询符合的，例如：q=mm&fq=date_time:[20081001 TO 20091031]，找关键字 mm，并且 date_time 是 20081001 到 20091031 之间的。

q.op – 覆盖 schema.xml 的 defaultOperator（有空格时用”AND”还是用”OR”操作逻辑），一般默认指定

df – 默认的查询字段，一般默认指定

qt – （ query type ）指定那个类型来处理查询请求，一般不用指定，默认是 standard。

- 排除在要排除的词前加上 “-” (不包含”号) 号

其它

indent – 返回的结果是否缩进，默认关闭，用 indent=true|on 开启，一般调试 json,php,phps,ruby 输出才有必要用这个参数。

version – 查询语法的版本，建议不使用它，由服务器指定默认值。

[Solr 的检索运算符]

“:” 指定字段查指定值，如返回所有值*:* ²

“?”²表示单个任意字符的通配

“*” 表示多个任意字符的通配（不能在检索的项开始使用*或者?符号） ²

“~”²表示模糊检索，如检索拼写类似于”roam”的项这样写：roam~将找到形如 foam 和 roams 的单词；roam~0.8，检索返回相似度在 0.8 以上的记录。

邻近检索，如检索相隔 10 个单词的”apache”和”jakarta”，”jakarta apache”~10

“^”²控制相关度检索，如检索 jakarta apache，同时希望去让”jakarta”的相关度更加好，那么在其后加上”^”符号和增量值，即 jakarta^4 apache

布尔操作符 AND、|| ²

布尔操作符 OR、 &&

布尔操作符 NOT、!、- ²（排除操作符不能单独与项使用构成查询）

“+” 存在操作符，要求符号”+”后的项必须在文档相应的域中存在 ²

( ) 用于构成子查询 ²

²[] 包含范围检索，如检索某时间段记录，包含头尾，date:[200707 TO 200710]

{} 不包含范围检索，如检索某时间段记录，不包含头尾

date:{200707 TO 200710}

” 转义操作符，特殊字符包括+ – & | ! ( ) { } [ ] ^ ” ~ * ? : “

二.高亮

hl-highlight，h1=true，表示采用高亮。可以用 h1.fl=field1,field2 来设定高亮显示的字段。

- hl.fl: 用空格或逗号隔开的字段列表。要启用某个字段的 highlight 功能，就得保证该字段在 schema 中是 stored。如果该参数未被给出，那么就会高 亮默认字段 standard handler 会用 df 参数，dismax 字段用 qf 参数。你可以使用星号去方便的高亮所有字段。如果你使用了通配符，那么要考虑启用 hl.requiredFieldMatch 选项。

- hl.requireFieldMatch:
  如果置为 true，除非该字段的查询结果不为空才会被高亮。它的默认值是 false，意味 着它可能匹配某个字段却高亮一个不同的字段。如果 hl.fl 使用了通配符，那么就要启用该参数。尽管如此，如果你的查询是 all 字段（可能是使用 copy-field 指令），那么还是把它设为 false，这样搜索结果能表明哪个字段的查询文本未被找到
- hl.usePhraseHighlighter:
  如果一个查询中含有短语（引号框起来的）那么会保证一定要完全匹配短语的才会被高亮。
- hl.highlightMultiTerm
  如果使用通配符和模糊搜索，那么会确保与通配符匹配的 term 会高亮。默认为 false，同时 hl.usePhraseHighlighter 要为 true。
- hl.snippets：
  这是 highlighted 片段的最大数。默认值为 1，也几乎不会修改。如果某个特定的字段的该值被置为 0（如 f.allText.hl.snippets=0），这就表明该字段被禁用高亮了。你可能在 hl.fl=*时会这么用。
- hl.fragsize:
  每个 snippet 返回的最大字符数。默认是 100.如果为 0，那么该字段不会被 fragmented 且整个字段的值会被返回。大字段时不会这么做。
- hl.mergeContiguous:
  如果被置为 true，当 snippet 重叠时会 merge 起来。
- hl.maxAnalyzedChars:
  会搜索高亮的最大字符，默认值为 51200，如果你想禁用，设为-1
- hl.alternateField:
  如果没有生成 snippet（没有 terms 匹配），那么使用另一个字段值作为返回。
- hl.maxAlternateFieldLength:
  如果 hl.alternateField 启用，则有时需要制定 alternateField 的最大字符长度，默认 0 是即没有限制。所以合理的值是应该为
- hl.snippets * hl.fragsize 这样返回结果的大小就能保持一致。
- hl.formatter:一个提供可替换的 formatting 算法的扩展点。默认值是 simple，这是目前仅有的选项。
- 显然这不够用，你可以看看 org.apache.solr.highlight.HtmlFormatter.java 和 solrconfig.xml 中 highlighting 元素是如何配置的。
  注意在不论原文中被高亮了什么值的情况下，如预先已存在的 em tags，也不会被转义，所以在有时会导致假的高亮。
- hl.fragmenter:
  这个是 solr 制 定 fragment 算法的扩展点。gap 是默认值。regex 是另一种选项，这种选项指明 highlight 的边界由一个正则表达式确定。这是一种非典型 的高级选项。为了知道默认设置和 fragmenters (and formatters)是如何配置的，可以看看 solrconfig.xml 中的 highlight 段。
  regex 的 fragmenter 有如下选项：
- hl.regex.pattern:正则表达式的 pattern
- hl.regex.slop:这是 hl.fragsize 能变化以适应正则表达式的因子。默认值是 0.6，意思是如果 hl.fragsize=100 那么 fragment 的大小会从 40-160.

三.分组查询：

1. Field Facet

Facet 字段通过在请求中加入 ” facet.field” 参数加以声明，如果需要对多个字段进行 Facet 查询，那么将该参数声明多次 . 比如

```
/select?q=联想

&facet=on
```

```
&facet.field=cpu
&facet.field=videoCard
```

各个 Facet 字段互不影响，且可以针对每个 Facet 字段设置查询参数 . 以下介绍的参数既可以应用于所有的 Facet 字段，也可以应用于每个单独的 Facet 字段 . 应用于单独的字段时通过

```
f.字段名.参数名=参数值
```

这种方式调用 . 比如 facet.prefix 参数应用于 cpu 字段，可以采用如下形式

```
f.cpu.facet.prefix=Intel
```

1.1 facet.prefix

表示 Facet 字段值的前缀 . 比如 "facet.field=cpu&facet.prefix=Intel", 那么对 cpu 字段进行 Facet 查询，返回的 cpu 都是以 "Intel" 开头的，" AMD" 开头的 cpu 型号将不会被统计在内 .

1.2 facet.sort

表示 Facet 字段值以哪种顺序返回 . 可接受的值为 true(count)|false(index,lex). true(count) 表示按照 count 值从大到小排列 . false(index,lex) 表示按照字段值的自然顺序 (字母，数字的顺序) 排列 . 默认情况下为 true(count). 当 facet.limit 值为负数时 ,默认 facet.sort= false(index,lex).

1.3 facet.limit

限制 Facet 字段返回的结果条数 . 默认值为 100. 如果此值为负数，表示不限制 .

1.4 facet.offset

返回结果集的偏移量，默认为 0. 它与 facet.limit 配合使用可以达到分页的效果 .

1.5 facet.mincount

限制了 Facet 字段值的最小 count, 默认为 0. 合理设置该参数可以将用户的关注点集中在少数比较热门的领域 .

1.6 facet.missing

默认为 " "，如果设置为 true 或者 on, 那么将统计那些该 Facet 字段值为 null 的记录.

1.7 facet.method

取值为 enum 或 fc, 默认为 fc. 该字段表示了两种 Facet 的算法，与执行效率相关 .

enum 适用于字段值比较少的情况，比如字段类型为布尔型，或者字段表示中国的所有省份.Solr 会遍历该字段的所有取值，并从 filterCache 里为每个值分配一个 filter(这里要求 solrconfig.xml 里对 filterCache 的设置足够大). 然后计算每个 filter 与主查询的交集 .

fc(表示 Field Cache) 适用于字段取值比较多，但在每个文档里出现次数比较少的情况 .Solr 会遍历所有的文档，在每个文档内搜索 Cache 内的值，如果找到就将 Cache 内该值的 count 加 1.

1.8 facet.enum.cache.minDf

当 facet.method=enum 时，此参数其作用 ,minDf 表示 minimum document frequency. 也就是文档内出现某个关键字的最少次数 . 该参数默认值为 0. 设置该参数可以减少 filterCache 的内存消耗，但会增加总的查询时间 (计算交集的时间增加了). 如果设置该值的话 ,官方文档建议优先尝试 25-50 内的值 .

2. Date Facet

日期类型的字段在文档中很常见，如商品上市时间，货物出仓时间，书籍上架时间等等．某些情况下需要针对这些字段进行 Facet. 不过时间字段的取值有无限性，用户往往关心的不是某个时间点而是某个时间段内的查询统计结果．Solr 为日期字段提供了更为方便的查询统计方式 .当然，字段的类型必须是 DateField(或其子类型).

需要注意的是，使用 Date Facet 时，字段名，起始时间，结束时间，时间间隔这 4 个参数都必须提供．

与 Field Facet 类似 ,Date Facet 也可以对多个字段进行 Facet. 并且针对每个字段都可以单独设置参数．

2.1 facet.date

该参数表示需要进行 Date Facet 的字段名，与 facet.field 一样，该参数可以被设置多次，表示对多个字段进行 Date Facet.

2.2 facet.date.start

起始时间，时间的一般格式为＂ 1995-12-31T23:59:59Z＂，另外可以使用＂ NOW＂,＂ YEAR＂,＂ MONTH＂ 等等，具体格式可以参考 org.apache.solr.schema. DateField 的 java doc.

2.3 facet.date.end

结束时间．

2.4 facet.date.gap

时间间隔．如果 start 为 2009-1-1,end 为 2010-1-1.gap 设置为＂ +1MONTH＂ 表示间隔 1 个月，那么将会把这段时间划分为 12 个间隔段．注意＂ +＂ 因为是特殊字符所以应该用＂ %2B＂ 代替．

2.5 facet.date.hardend

取值可以为 true|false, 默认为 false. 它表示 gap 迭代到 end 处采用何种处理．举例说明 start 为 2009-1-1,end 为 2009-12-25,gap 为＂ +1MONTH＂,hardend 为 false 的话最后一个时间段为 2009-12-1 至 2010-1-1;hardend 为 true 的话最后一个时间段为 2009-12-1 至 2009-12-25.

2.6 facet.date.other

取值范围为 before|after|between|none|all, 默认为 none.

before 会对 start 之前的值做统计．

after 会对 end 之后的值做统计．

between 会对 start 至 end 之间所有值做统计．如果 hardend 为 true 的话，那么该值就是各个时间段统计值的和．

none 表示该项禁用．

all 表示 before,after,all 都会统计．

举例：

```
&facet=on
&facet.date=date
&facet.date.start=2009-1-1T0:0:0Z
&facet.date.end=2010-1-1T0:0:0Z
&facet.date.gap=%2B1MONTH
&facet.date.other=all
```

返回结果：

```
<lst name="facet_counts">
        <lst name="facet_queries"/>
        <lst name="facet_fields"/>
        <lst name="facet_dates">
<int name="2009-01-01T00:00:00Z">5</int>
<int name="2009-11-01T00:00:00Z">1</int>
<int name="2009-12-01T00:00:00Z">5</int>
<str name="gap">+1MONTH</str>
<date name="end">2010-01-01T00:00:00Z</date>
<int name="before">180</int>
<int name="after">5</int>
<int name="between">54</int>
</lst>
</lst>
```

3. Facet Query

Facet Query 利用类似于 filter query 的语法提供了更为灵活的 Facet. 通过 facet.query 参数 , 可以对任意字段进行筛选 .

例 1:

```
&facet=on
&facet.query=date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]
&facet.query=date:[2009-4-1T0:0:0Z TO 2009-5-1T0:0:0Z]
```

返回结果 :

```
<lst name="facet_counts">
        <lst name="facet_queries">
                <int name="date:[2009-1-1T0:0:0Z TO
2009-2-1T0:0:0Z]">5</int>
<int name="date:[2009-4-1T0:0:0Z TO 2009-5-1T0:0:0Z]">3</int>
</lst>
        <lst name="facet_fields"/>
        <lst name="facet_dates"/>
</lst>
```

例 2:

```
&facet=on
&facet.query=date:[2009-1-1T0:0:0Z TO 2009-2-1T0:0:0Z]
&facet.query=price:[* TO 5000]
```

返回结果 :

```
<lst name="facet_counts">
        <lst name="facet_queries">
                <int name="date:[2009-1-1T0:0:0Z TO
2009-2-1T0:0:0Z]">5</int>
<int name="price:[* TO 5000]">116</int>
</lst>
        <lst name="facet_fields"/>
```

```
        <lst name="facet_dates"/>
</lst>
```

例 3:

```
&facet=on
&facet.query=cpu:[A TO G]
```

返回结果 :

```
<lst name="facet_counts">
        <lst name="facet_queries">
                <int name="cpu:[A TO G]">11</int>
</lst>
        <lst name="facet_fields"/>
        <lst name="facet_dates"/>
</lst>
```

4. key 操作符

可以用 key 操作符为 Facet 字段取一个别名 .

例 :

```
&facet=on
&facet.field={!key=中央处理器}cpu
&facet.field={!key=显卡}videoCard
```

返回结果 :

```
<lst name="facet_counts">
        <lst name="facet_queries"/>
```

```
        <lst name="facet_fields">
                <lst name="中央处理器">
                        <int name="Intel 酷睿 2 双核 T6600">48</int>
                        <int name="Intel 奔腾双核 T4300">28</int>

                </lst>
                <lst name="显卡">
                        <int name="ATI Mobility Radeon HD 4">63</int>
                        <int name="NVIDIA GeForce G 105M">24</int>
<int name="NVIDIA GeForce GT 240M">21</int>
<int name="NVIDIA GeForce G 103M">8</int>
<int name="NVIDIA GeForce GT 220M">8</int>
<int name="NVIDIA GeForce 9400M G">7</int>
<int name="NVIDIA GeForce G 210M">6</int>
</lst>
        </lst>
        <lst name="facet_dates"/>
</lst>
```

5. tag 操作符和 ex 操作符

当查询使用 filter query 的时候，如果 filter query 的字段正好是 Facet 字段，那么查询结果往往被限制在某一个值内 .

例 :

```
&fq=screenSize:14
&facet=on
&facet.field=screenSize
```

返回结果：

```
<lst name="facet_counts">
        <lst name="facet_queries"/>
        <lst name="facet_fields">
                <lst name=" screenSize">
                        <int name="14.0">107</int>
<int name="10.2">0</int>
<int name="11.1">0</int>
</lst>
        </lst>
        <lst name="facet_dates"/>
</lst>
```

可以看到，屏幕尺寸 (screenSize) 为 14 寸的产品共有 107 件，其它尺寸的产品的数目都是 0, 这是因为在 filter 里已经限制了 screenSize:14. 这样，查询结果中，除了 screenSize=14 的这一项之外，其它项目没有实际的意义 .

有些时候，用户希望把结果限制在某一范围内，又希望查看该范围外的概况 . 比如上述情况 ,既要把查询结果限制在 14 寸屏的笔记本，又想查看一下其它屏幕尺寸的笔记本有多少产品 . 这个时候需要用到 tag 和 ex 操作符 .

tag 就是把一个 filter 标记起来 ,ex(exclude) 是在 Facet 的时候把标记过的 filter 排除在外 .

例：

```
&fq={!tag=aa}screenSize:14
&facet=on
```

```
&facet.field={!ex=aa}screenSize
```

返回结果：

```
<lst name="facet_counts">
        <lst name="facet_queries"/>
        <lst name="facet_fields">
                <lst name=" screenSize">
                        <int name="14.0">107</int>
<int name="14.1">40</int>
<int name="13.3">34</int>
</lst>
        </lst>
        <lst name="facet_dates"/>
</lst>
```